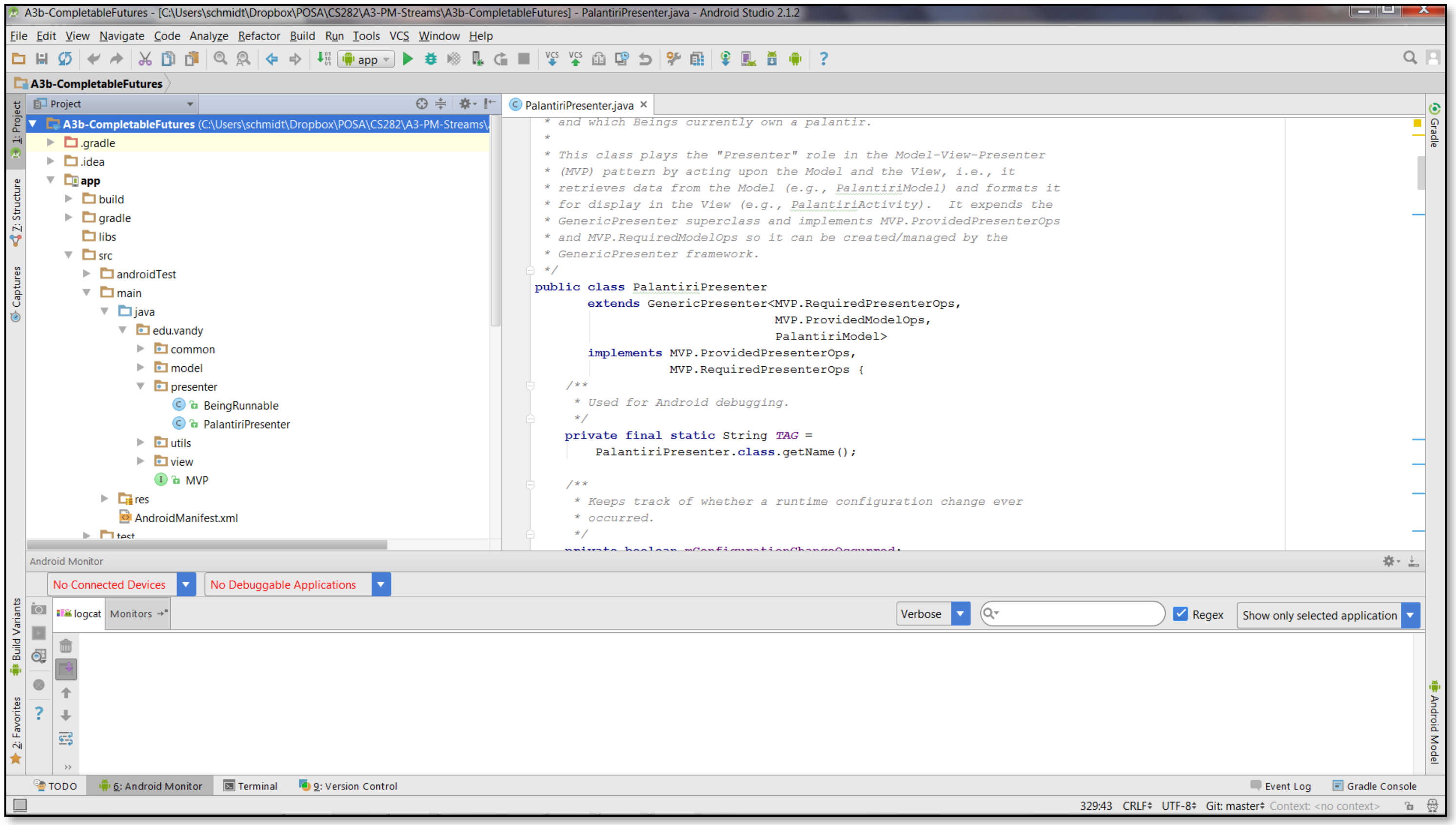
The Composite Pattern

Implementation in C++

Douglas C. Schmidt

Learning Objectives in This Lesson

- Recognize how the *Composite* pattern can be applied to make the expression tree more uniform & extensible.
- Understand the structure & functionality of the *Composite* pattern.
- Know how to implement the *Composite* pattern in C++.



A3b-CompletableFutures - [C:\Users\schmidt\Dropbox\POSA\CS282\A3-PM-Streams\A3b-CompletableFutures] - PalantiriPresenter.java - Android Studio 2.1.2

PalantiriPresenter.java

```
 * and which Beings currently own a palantir.
 *
 * This class plays the "Presenter" role in the Model-View-Presenter
 * (MVP) pattern by acting upon the Model and the View, i.e., it
 * retrieves data from the Model (e.g., PalantiriModel) and formats it
 * for display in the View (e.g., PalantiriActivity).  It expends the
 * GenericPresenter superclass and implements MVP.ProvidedPresenterOps
 * and MVP.RequiredModelOps so it can be created/managed by the
 * GenericPresenter framework.
 */
public class PalantiriPresenter
       extends GenericPresenter<MVP.RequiredPresenterOps,
                                MVP.ProvidedModelOps,
                                PalantiriModel>
       implements MVP.ProvidedPresenterOps,
                  MVP.RequiredPresenterOps {
    /**
     * Used for Android debugging.
     */
    private final static String TAG =
        PalantiriPresenter.class.getName();

    /**
     * Keeps track of whether a runtime configuration change ever
     * occurred.
     */
```

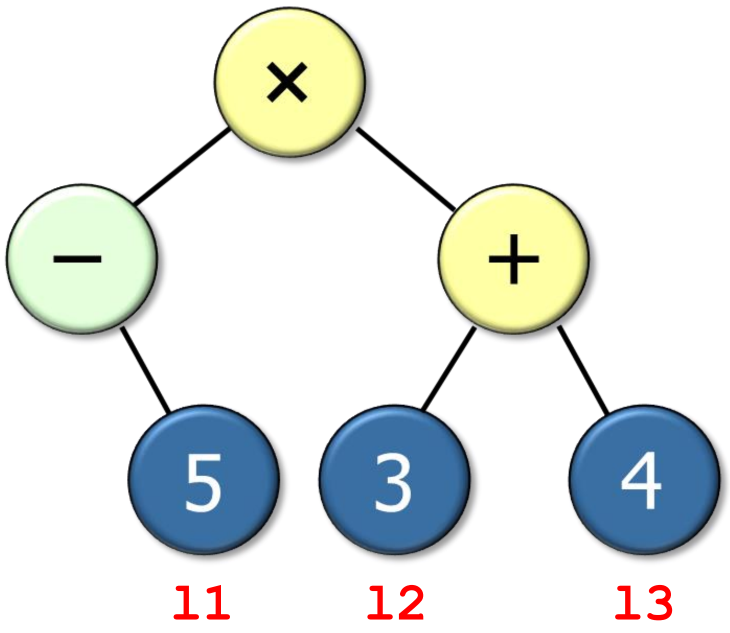
Douglas C. Schmidt

Implementing the Composite Pattern in C++

Composite example in C++

- Build an expression tree based on recursively composed objects.

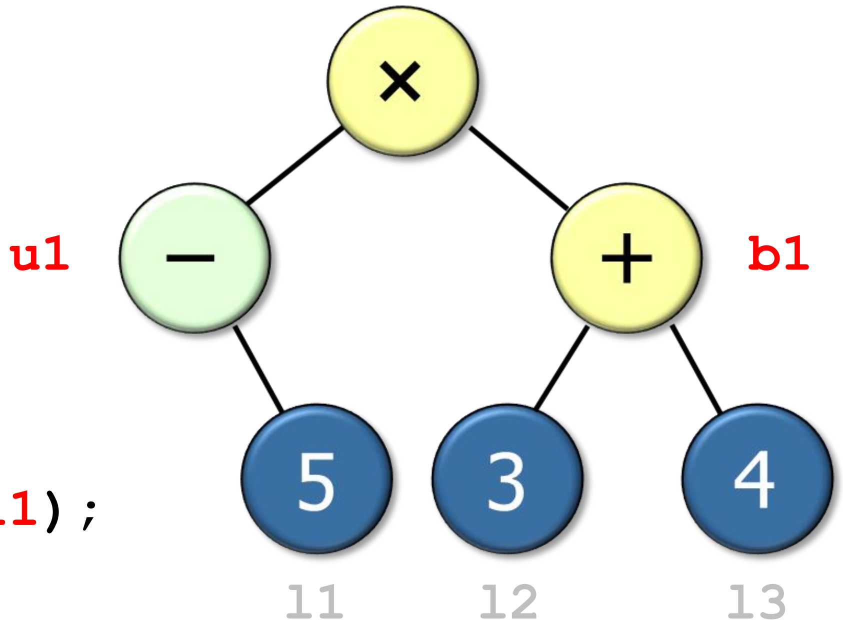
```
Component_Node *l1 =
  new Leaf_Node(5);
Component_Node *l2 =
  new Leaf_Node(3);
Component_Node *l3 =
  new Leaf_Node(4);
```



Composite example in C++

- Build an expression tree based on recursively composed objects.

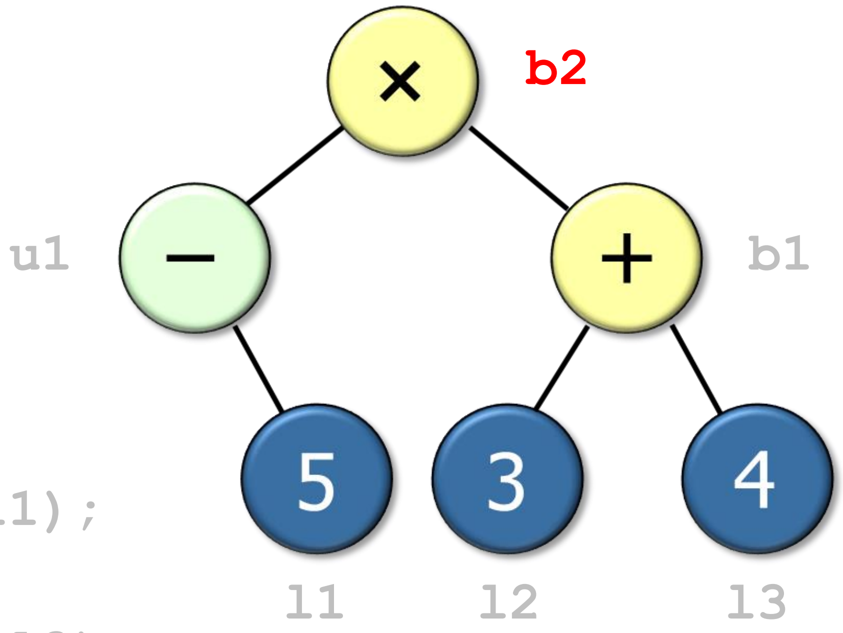
```
Component_Node *l1 =
  new Leaf_Node(5);
Component_Node *l2 =
  new Leaf_Node(3);
Component_Node *l3 =
  new Leaf_Node(4);
Component_Node *u1 =
  new Composite_Negate_Node(l1);
Component_Node *b1 =
  new Composite_Add_Node(l2, l3);
```



Composite example in C++

- Build an expression tree based on recursively composed objects.

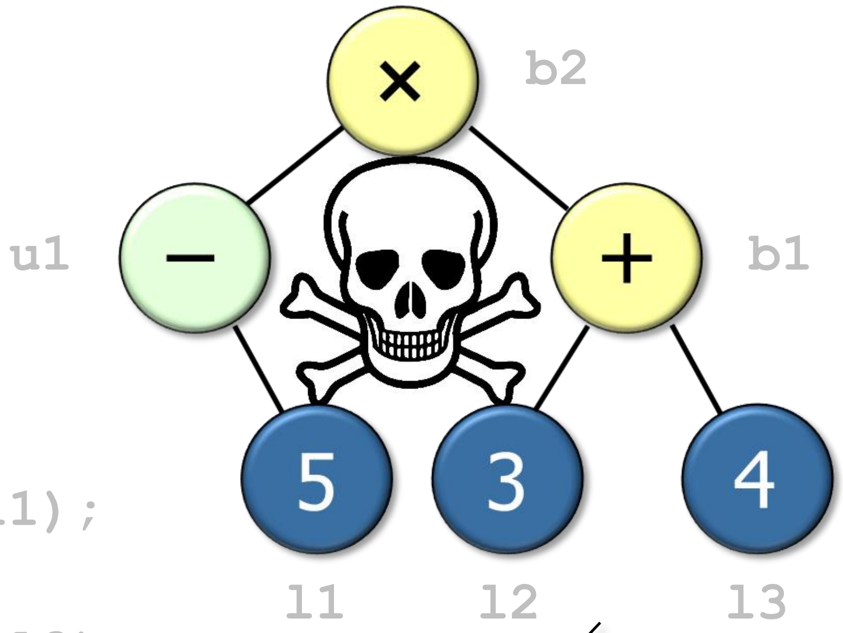
```
Component_Node *l1 =
  new Leaf_Node(5);
Component_Node *l2 =
  new Leaf_Node(3);
Component_Node *l3 =
  new Leaf_Node(4);
Component_Node *u1 =
  new Composite_Negate_Node(l1);
Component_Node *b1 =
  new Composite_Add_Node(l2, l3);
Component_Node *b2 =
  new Composite_Multiply_Node(u1, b1);
```



Composite example in C++

- Build an expression tree based on recursively composed objects.

```
Component_Node *l1 =
  new Leaf_Node(5);
Component_Node *l2 =
  new Leaf_Node(3);
Component_Node *l3 =
  new Leaf_Node(4);
Component_Node *u1 =
  new Composite_Negate_Node(l1);
Component_Node *b1 =
  new Composite_Add_Node(l2, l3);
Component_Node *b2 =
  new Composite_Multiply_Node(u1, b1);
```



C++ lacks garbage collector, so this implementation requires intentional de-allocation of memory (e.g., via reference counting) when it's no longer needed.

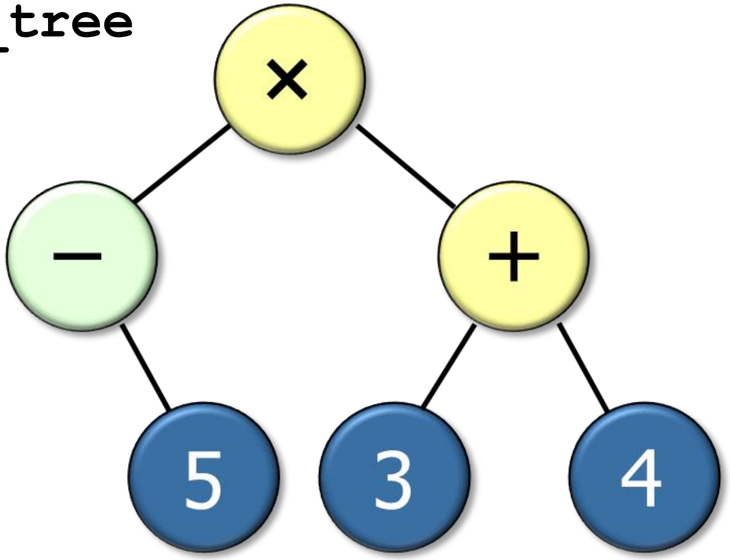
See www.linuxprogrammingblog.com/cpp-objects-reference-counting

Composite example in C++

- Build an expression tree based on recursively composed objects.

```
unique_ptr<Component_Node> expr_tree
  = make_expression_tree
       ("-5 * (3 + 4)");
```

A better way to build an expression tree is to apply a Creational pattern that shields client programs from the details of how the composite expression tree is implemented



See upcoming lessons on the *Builder* (& *Interpreter*) patterns.